(12) INTERNATIONAL APPLICATION PUBLISHED UNDER THE PATENT COOPERATION TREATY (PCT)

(19) World Intellectual Property Organization
International Bureau

(43) International Publication Date
22 August 2002 (22.08.2002)

PCT

(10) International Publication Number
WO 02/065258 A2

(51) International Patent Classification[7]: G06F 1/00

(21) International Application Number: PCT/US01/04834

(22) International Filing Date: 13 February 2001 (13.02.2001)

(25) Filing Language: English

(26) Publication Language: English

(71) Applicant: QUALCOMM INCORPORATED [US/US]; 5775 Morhouse Drive, San Diego, CA 92121-1714 (US).

(72) Inventors: JOHNSON, Paul, K.; 1284 Summit Place, Escondido, CA 92025-7561 (US). QUICK, Roy, F., Jr.; 4502 Del Monte Avenue, San Diego, CA 92126 (US).

(74) Agents: WADSWORTH, Philip, R. et al.; Qualcomm Incorporated, 5775 Morehouse Drive, San Diego, CA 92121-1714 (US).

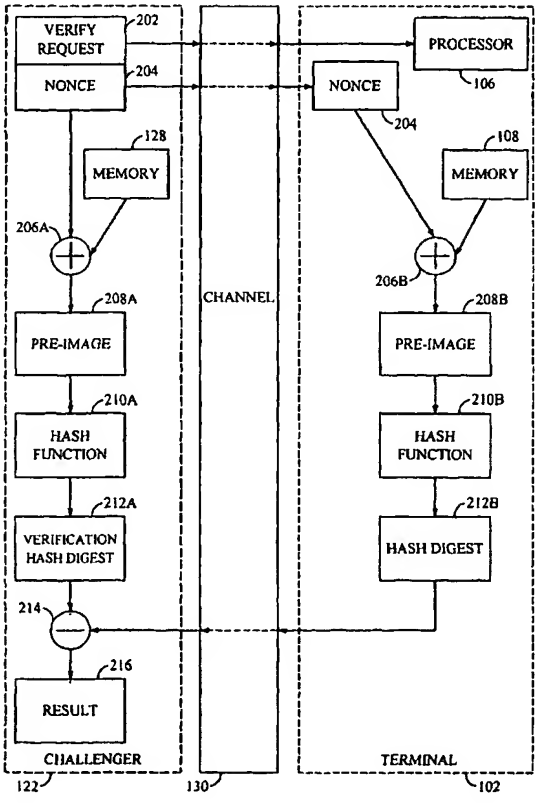
(81) Designated States *(national)*: AE, AG, AL, AM, AT, AU, AZ, BA, BB, BG, BR, BY, BZ, CA, CH, CN, CR, CU, CZ, DE, DK, DM, DZ, EE, ES, FI, GB, GD, GE, GH, GM, HR, HU, ID, IL, IN, IS, JP, KE, KG, KP, KR, KZ, LC, LK, LR, LS, LT, LU, LV, MA, MD, MG, MK, MN, MW, MX, MZ, NO, NZ, PL, PT, RO, RU, SD, SE, SG, SI, SK, SL, TJ, TM, TR, TT, TZ, UA, UG, UZ, VN, YU, ZA, ZW.

(84) Designated States *(regional)*: ARIPO patent (GH, GM, KE, LS, MW, MZ, SD, SL, SZ, TZ, UG, ZW), Eurasian patent (AM, AZ, BY, KG, KZ, MD, RU, TJ, TM), European patent (AT, BE, CH, CY, DE, DK, ES, FI, FR, GB, GR, IE, IT, LU, MC, NL, PT, SE, TR), OAPI patent (BF, BJ, CF, CG, CI, CM, GA, GN, GW, ML, MR, NE, SN, TD, TG).

Published:
— *without international search report and to be republished upon receipt of that report*

[Continued on next page]

(54) Title: METHOD AND APPARATUS FOR AUTHENTICATING EMBEDDED SOFTWARE IN A REMOTE UNIT OVER A COMMUNICATIONS CHANNEL



(57) Abstract: A method, apparatus, and computer program product for authenticating embedded software in the memory of a responder over an unprotected channel. The method includes the steps of transmitting a verify request and a unique nonce form a challenger to the responder over the unprotected channel; processing the embedded software and the nonce using a cryptographic hash function to produce a hash digest, wherein the embedded software includes a unique identifier; transmitting the hash digest to the challenger; processing a copy of the embedded software and the nonce using the cryptographic hash function to produce a verification hash digest; and authenticating the embedded software when the received hash digest and the verification hash digest match.

WO 02/065258 A2

For two-letter codes and other abbreviations, refer to the "Guidance Notes on Codes and Abbreviations" appearing at the beginning of each regular issue of the PCT Gazette.

METHOD AND APPARATUS FOR AUTHENTICATING EMBEDDED SOFTWARE IN A REMOTE UNIT OVER A COMMUNICATIONS CHANNEL

BACKGROUND OF THE INVENTION

I. Field of the Invention

The present invention relates to communication systems. More specifically, the present invention relates to authenticating embedded software in a remote unit.

II. Related Art

Wireless communication networks are enjoying notable popularity in all aspects of business, industry and personal life. As such, portable, hand-held communication devices have experienced widespread growth in recent years. Portable devices such as cellular phones are now commonplace with business and personal users alike. Additionally, advanced systems, such as satellite communications systems using portable, hand-held and mobile phones, are on the horizon.

Such portable communications devices, referred to herein as "user terminals" or simply "terminals," usually communicate with a base station over an air link. In many situations it is desirable for the base station to ascertain the identity of a particular user terminal. This process is referred to as "authenticating" the user terminal. One such situation is where secure communications with a user terminal are required. Authenticating the user terminal ensures that an "impostor" user terminal has not been substituted for a legitimate user terminal.

Further, it is desirable to ascertain the version of the software executing within the user terminal. A secure user terminal (that is, one designed for secure communications) usually contains a read-only memory (ROM) that contains boot software that is guaranteed to execute when the phone is turned on. A saboteur could thwart this system by simply substituting a ROM

containing impostor boot software for the ROM containing legitimate boot software. Authenticating the software within the ROM ensures that the proper boot software has been executed to secure the link.

Authenticating the user terminal embedded software ensures that the user's secure communications capability is intact. Authenticating both the user terminal and the version of the embedded software in the user terminal's memory is useful, for example, in determining whether to download a software upgrade to the user terminal. In this example, authenticating the version of the embedded software can prevent a user from obtaining a software upgrade that was purchased by another user.

SUMMARY OF THE INVENTION

The present invention is a method, apparatus, and computer program product for authenticating embedded software in the memory of a responder over an unprotected channel. The method includes the steps of transmitting a verify request and a unique nonce from a challenger to the terminal over the unprotected channel; processing the embedded software and the nonce using a cryptographic hash function to produce a hash digest, wherein the embedded software includes a unique identifier; transmitting the hash digest to the challenger; processing a copy of the embedded software and the nonce using the cryptographic hash function to produce a verification hash digest; and authenticating the embedded software when the received hash digest and the verification hash digest match.

The present invention is also directed to a responder that includes a processor that processes the embedded software to produce a digest and a transmitter that transmits the digest to the challenger, whereby the challenger can authenticate the embedded software using the digest and a verification digest produced by processing a copy of the embedded software.

The present invention is also directed to a challenger that includes a receiver that receives a digest from the terminal, the digest produced by processing the embedded software; and a processor that processes the received

digest and a verification digest to produce a result, whereby the embedded software is authenticated when the result indicates a match.

One advantage of the present invention is that it authenticates the identity of a remote terminal over an unprotected channel.

Another advantage of the present invention is that it authenticates the version of embedded software resident in the memory of a remote terminal over an unprotected channel.

BRIEF DESCRIPTION OF THE FIGURES

The features, objects, and advantages of the present invention will become more apparent from the detailed description set forth below when taken in conjunction with the drawings in which like reference characters identify corresponding elements throughout and wherein:

FIG. 1 is a block diagram of a communications system according to a preferred embodiment of the present invention.

FIG. 2 is a flow diagram describing the operation of the present invention according to a preferred embodiment.

FIG. 3 is a flowchart describing the operation of the present invention according to a preferred embodiment.

FIG. 4 is an exemplary computer system capable of carrying out the functionality of the present invention.

DETAILED DESCRIPTION OF THE PREFERRED EMBODIMENTS

The present invention is a method and apparatus for authenticating embedded software in a remote unit over a communication channel.

Example Environment

Before describing the invention in great detail, it is useful to describe an example environment in which the invention can be implemented. The present invention can be implemented in any communication system, and is especially useful in communication systems having unprotected communication channels. As used herein, the term "communication channel" includes any medium used for transmission of a signal, including hard-wired, wireless, optic fiber, and the like. An "unprotected" channel is one that does not guarantee that messages cannot be modified during transit over the channel. The above-described environments include, without limitation, cellular communication systems, personal communication systems, satellite communication systems, and many others.

In addition, the present invention is especially useful in verifying the contents of a memory in a remote unit. These contents are often referred to as "embedded software." One example of embedded software is the executable code residing in the ROM of a cellular telephone. In that example, the present invention can be used to authenticate the embedded software over the communications link between the cellular telephone and a base station. Another example of embedded software is the Basic Input/Output System (BIOS) of a personal computer. In that example, the present invention could be used to authenticate the BIOS of a personal computer over a modem link or over the Internet.

Preferred Embodiments

FIG. 1 is a block diagram of a communications system 100 according to a preferred embodiment of the present invention. The system includes a

challenger 122 and a responder 102 which communicate over a communications channel 130. Channel 130 can be an unprotected communications channel. However, this is not required by the present invention.

Responder 102 includes a transceiver 104, a processor 106 and a memory 108. Transceiver 104 permits responder 102 to communicate over channel 130. In a preferred embodiment, memory 108 includes a flash memory 110 and a boot block 112. Flash memory 110 can be any memory that is in-circuit programmable (that is, programmable while mounted within responder 102). Boot block 112 can be any memory that is not in-circuit programmable, such as a read-only memory (ROM).

Challenger 122 includes a transceiver 124 capable of communicating over channel 130 and a processor 126 capable of performing the functions of challenger 122 described herein. In a preferred embodiment, challenger 122 also includes a memory 128.

Challenger 122 and responder 102 can reside within any two communications devices that communicate over a communications channel. For example, in a cellular telephone system, challenger 122 could be located at a cellular base station and responder 102 could be part of a cellular telephone. In another example, responder 102 could be a satellite telephone at the end of a manufacturing assembly line and challenger 122 could be a test unit verifying the identity of the responder and its software. Many other example implementations exist.

According to one embodiment of the present invention, challenger 122 transmits a verify request to responder 102 over unprotected channel 130. In response, responder 102 processes the embedded software using a hash function to produce a hash digest, as described in detail below. Responder 102 transmits the hash digest to challenger 122 over channel 130. Challenger 122 processes the received hash digest and a verification hash digest to produce a result. Challenger 122 then authenticates the embedded software within responder 102 according to this result.

A hash function is a function that converts a variable-length input string, called a pre-image, to a fixed-length output string, called a hash digest, which is

generally smaller than the pre-image. One example of a hash function is a simple calculation of the exclusive-or of all of the bytes of the pre-image to produce a one-byte hash digest.

The purpose of the hash function is to "fingerprint" the pre-image. In other words, the purpose is to produce a value that indicates whether a candidate pre-image is likely to be the same as a known pre-image. The hash function itself is known. The security of a hash function results from the fact that the process is not reversible. That is, the hash digest is not dependent on the pre-image in any discernable way. Given a hash digest, it is computationally unfeasible to find the pre-image that generated that digest. However, a hash digest is ideal for comparing two pre-images to determine whether they are identical. In general, a single-bit change in a pre-image changes approximately half of the bits in the resulting hash digest.

However, under certain conditions, changes in the pre-image may not affect the hash digest for certain hash functions. One such situation is where a pre-image contains little data and a large amount of single-value fill data (for example, a fill of all zeroes). A class of hash functions has been developed to remedy this situation. These hash functions are called "cryptographic" hash functions. One such function is the well-known SHA-1 secure hash algorithm. In a preferred embodiment, a cryptographic hash function is used to process the embedded software data.

In another embodiment, the verify request is accompanied by a value referred to as a "nonce." A nonce is a value generated by the challenger for use in challenging a responder. In one embodiment, a unique nonce is used. A unique nonce is a value used no more than once for the same purpose. Responder 102 processes the nonce and the embedded software using a hash function to produce the hash digest. In a preferred embodiment, the nonce is unique to each challenge of the responder. Therefore, this process produces a different hash digest for each challenge of a given responder. Thus, an impostor responder cannot defeat the authentication merely by transmitting a hash digest that was successfully used by a legitimate responder. This use of a nonce thereby serves to increase the reliability of the authentication.

In another embodiment, the embedded software includes an identifier that uniquely identifies the user terminal. In a preferred embodiment, the identifier is never transmitted over an unprotected channel. This prevents a saboteur from using an identifier for a legitimate terminal to emulate that terminal. In this embodiment, both the embedded software and the identity of the user terminal are authenticated.

FIGS. 2 and 3 are a flow diagram and a flowchart, respectively, describing the operation of the present invention according to a preferred embodiment. The process begins in step 302 when challenger 122 transmits a challenge, including a verify request and a nonce, to responder 102. Nonce 204 is a value that is created specifically for a particular challenge. In a preferred embodiment, nonce 204 is generated by challenger 122, and is not known to responder 102 prior to the challenge. Challenger 122 can transmit the challenge in a variety of ways. For example, in a cellular telephone system, the challenge can be transmitted to responder 102 over a paging channel or a traffic channel.

The verify request is received by processor 106, which processes nonce 204 and the embedded software stored in memory 108 using hash function 210B to produce hash digest 212B, as shown in step 310. In one embodiment, nonce 204 and the embedded software are catenated by catenator 206B to form a pre-image 208B for processing by hash function 210B. In a preferred embodiment, hash function 210B is a cryptographic hash function such as SHA-1. Of course, other methods can be used to produce a pre-image using the nonce and the embedded software without departing from the scope of the present invention, as would be apparent to one skilled in the relevant arts.

In one embodiment, the embedded software includes an identifier that uniquely identifies the user terminal. In this embodiment, both the embedded software and the identity of the user terminal are authenticated.

Pre-image 208B is processed using a hash function 210B to produce a hash digest 212B, as shown in step 310. Responder 102 then transmits hash digest 212B to challenger 122 over channel 130, as shown in step 312.

Challenger 122 processes the received hash digest 212B and a verification hash digest 212A to produce a result 216, as shown in step 306. In a preferred

embodiment, challenger 122 compares hash digest 212B and verification hash value 212A using difference element 214 to produce a result 216. The generation of verification hash digest 212A is discussed below.

Challenger 122 then authenticates the embedded software based on result 216. If result 216 indicates a match, then the embedded software is authenticated.

In a preferred embodiment, challenger 122 generates verification hash digest 212A using the same method that responder 102 uses to generate hash digest 212B. An exact copy of the embedded software stored in memory 108 of responder 102 is stored in memory 128 of challenger 122. In one embodiment, the copy of the embedded software includes the same identifier as the embedded software in the user terminal. In this embodiment, both the embedded software and the identity of the user terminal are authenticated.

Referring to FIGS. 2 and 3, challenger 122 processes nonce 204 and the copy of the embedded software, stored in memory 128 of challenger 122, using the same hash function as responder 102, to produce the verification hash digest 212A, as shown in step 304. In one embodiment, nonce 204 and the embedded software are catenated by catenator 206A to form a pre-image 208A for processing by hash function 210A. Of course, other methods can be used to produce a pre-image using the nonce and the embedded software without departing from the scope of the present invention, as would be apparent to one skilled in the relevant arts.

Computer Program Product

The present invention may be implemented using hardware, software or a combination thereof and may be implemented in a computer system or other processing system. In fact, in one embodiment, the invention is directed toward one or more computer systems capable of carrying out the functionality described herein. An example computer system 400 is shown in FIG. 4. The computer system 400 includes one or more processors, such as processor 404. The processor 404 is connected to a communication bus 406. Various software

embodiments are described in terms of this example computer system. After reading this description, it will become apparent to a person skilled in the relevant art how to implement the invention using other computer systems and/or computer architectures.

Computer system 400 also includes a main memory 408, preferably random access memory (RAM), and can also include a secondary memory 410. The secondary memory 410 can include, for example, a hard disk drive 412 and/or a removable storage drive 414, representing a floppy disk drive, a magnetic tape drive, an optical disk drive, etc. The removable storage drive 414 reads from and/or writes to a removable storage unit 418 in a well known manner. Removable storage unit 418, represents a floppy disk, magnetic tape, optical disk, etc. which is read by and written to by removable storage drive 414. As will be appreciated, the removable storage unit 418 includes a computer usable storage medium having stored therein computer software and/or data.

In alternative embodiments, secondary memory 410 may include other similar means for allowing computer programs or other instructions to be loaded into computer system 400. Such means can include, for example, a removable storage unit 422 and an interface 420. Examples of such include a program cartridge and cartridge interface (such as that found in video game devices), a removable memory chip (such as an EPROM, or PROM) and associated socket, and other removable storage units 422 and interfaces 420 which allow software and data to be transferred from the removable storage unit 418 to computer system 400.

Computer system 400 can also include a communications interface 424. Communications interface 424 allows software and data to be transferred between computer system 400 and external devices. Examples of communications interface 424 can include a modem, a network interface (such as an Ethernet card), a communications port, a PCMCIA slot and card, etc. Software and data transferred via communications interface 424 are in the form of signals which can be electronic, electromagnetic, optical or other signals capable of being received by communications interface 424. These signals 426

are provided to communications interface 424 via a channel 428. This channel 428 carries signals 426 and can be implemented using wire or cable, fiber optics, a phone line, a cellular phone link, an RF link and other communications channels.

In this document, the terms "computer program medium" and "computer usable medium" are used to generally refer to media such as removable storage device 418, a hard disk installed in hard disk drive 412, and signals 426. These computer program products are means for providing software to computer system 400.

Computer programs (also called computer control logic) are stored in main memory 408 and/or secondary memory 410. Computer programs can also be received via communications interface 424. Such computer programs, when executed, enable the computer system 400 to perform the features of the present invention as discussed herein. In particular, the computer programs, when executed, enable the processor 404 to perform the features of the present invention. Accordingly, such computer programs represent controllers of the computer system 400.

In an embodiment where the invention is implemented using software, the software may be stored in a computer program product and loaded into computer system 400 using removable storage drive 414, hard drive 412 or communications interface 424. The control logic (software), when executed by the processor 404, causes the processor 404 to perform the functions of the invention as described herein.

In another embodiment, the invention is implemented primarily in hardware using, for example, hardware components such as application specific integrated circuits (ASICs). Implementation of the hardware state machine so as to perform the functions described herein will be apparent to persons skilled in the relevant art(s). In yet another embodiment, the invention is implemented using a combination of both hardware and software.

Memory Fill

Hash functions work best when supplied by varying data. Hash functions are weakened when the pre-image contains "empty space" populated by all ones, all zeros, or a repeating pattern. Such empty space occurs often when a memory, such as a ROM, is programmed with embedded software. ROMs are commercially available only in a few pre-determined capacities, such as one megabyte, two megabytes, and the like. Because the software is unlikely to fill such a ROM completely, empty space is likely to occur.

In a preferred embodiment of the present invention, the empty space within memory 108 of responder 102 is populated with a predetermined bit pattern, such as a random bit pattern. The pre-image for the hash function then includes the embedded software and the predetermined bit pattern. Such a varied pre-image increases the likelihood that no two hash digests will be the same. This process makes the responder's response to the challenger more difficult to emulate.

Example Implementations

The present invention is ideal for performing a software update for a remote terminal. In this embodiment, the updating authority (that is, challenger 122) has access to a copy of the contents of the memory 108 of responder 102, including the unique identifier associated with the terminal. These values are used as described above to authenticate the identity of responder 102 and to determine the version of the embedded software in responder memory 108.

The unique identifier is used by both challenger 122 and responder 102 in establishing a secure encryption key and/or in generating an initialization vector. The software update code is encrypted using the key and/or vector before it is sent to responder 102 over unprotected channel 130. This process guarantees that only the authenticated responder 102 receives the software update. Such a system is especially useful where commercial software updates are purchased on an individual terminal basis.

The present invention is also ideal for ensuring that a responder 102 is loaded with the proper software during its manufacture. At some point during the manufacturing process, memory 108 is loaded with the embedded software for the responder. The present invention can be used to verify that the proper software has been successfully loaded into the responder. Significantly, this test can occur over an unprotected channel on the factory floor.

Conclusion

The previous description of the preferred embodiments is provided to enable any person skilled in the art to make or use the present invention. The various modifications to these embodiments will be readily apparent to those skilled in the art, and the generic principles defined herein may be applied to other embodiments without the use of the inventive faculty. Thus, the present invention is not intended to be limited to the embodiments shown herein but is to be accorded the widest scope consistent with the principles and

What Is Claimed Is:

CLAIMS

1. A method for authenticating embedded software in the memory of a responder over a communications channel, comprising the steps of:

processing the embedded software to produce a digest;

transmitting said digest to a challenger;

processing the received digest and a verification digest to produce a result; and

authenticating the embedded software according to said result.

2. The method of claim 1, wherein the communications channel is unprotected.

3. The method of claim 2, further comprising the step of:

processing a copy of the embedded software to produce said verification digest.

4. The method of claim 3, further comprising the step of:

transmitting a verify request from said challenger to the responder over the channel.

5. The method of claim 4, wherein:

said step of processing the embedded software comprises the step of processing the embedded software using a hash function to produce said digest; and

said step of processing said copy of the embedded software comprises the step of processing said copy of the embedded software using said hash function to produce said verification digest.

6. The method of claim 5, further comprising the step of:

transmitting a nonce from said challenger to the responder; and wherein

said step of processing the embedded software comprises the step of processing said nonce and the embedded software using said hash function to produce said digest; and

said step of processing said copy of the embedded software comprises the step of processing said nonce and said copy of the embedded software using said hash function to produce said verification digest.

7. The method of claim 6, wherein said nonce is unique.

8. The method of claim 7, wherein

said step of processing the embedded software comprises the steps of

catenating said nonce and the embedded software to produce a pre-image, and

processing said pre-image using said hash function to produce said digest; and

said step of processing said copy of the embedded software comprises the steps of:

catenating said nonce and said copy of the embedded software to produce a verification pre-image, and

processing said verification pre-image using said hash function to produce said verification digest.

9. The method of claim 8, wherein the embedded software includes a unique identifier.

10. The method of claim 9, wherein:

said step of processing the embedded software includes the step of processing said nonce and the embedded software using a cryptographic hash function to produce said digest; and

said step of processing said copy of the embedded software includes the step of processing said nonce and said copy of the embedded software using said cryptographic hash function to produce said verification digest.

11. The method of claim 10, wherein said step of processing said received hash digest value and said verification hash digest value comprises the step of comparing said received digest and said verification digest.

12. The method of claim 11, wherein said authenticating step comprises the step of authenticating the embedded software when said received digest and said verification digest match.

13. The method of claim 12, wherein the embedded software occupies a portion of the memory of the responder, further comprising the step of:

populating the remaining portion of the memory with a predetermined bit pattern; and wherein

said step of processing said nonce and the embedded software comprises the step of processing said nonce, the embedded software and said predetermined bit pattern; and

said step of processing said nonce and said copy of the embedded software comprises the stop of processing said nonce, said copy of the embedded software and said predetermined bit pattern.

14. An apparatus for authenticating embedded software in the memory (108) of a responder (102) over a communications channel (130), comprising:

a responder processor (106) that processes the embedded software to produce a digest (212B);

a responder transceiver (104) that transmits said digest to a challenger (122);

a challenger processor (126) that processes said received digest and a verification digest (212A) to produce a result (216); and

means for authenticating (126) the embedded software according to said result.

15. The apparatus of claim 14, wherein the communications channel is unprotected.

16. The apparatus of claim 15, wherein said challenger processor comprises:

means for processing (126) a copy of the embedded software to produce said verification digest.

17. The apparatus of claim 16, further comprising:

a challenger transceiver (124) that transmits a verify request from said challenger to the responder over the channel.

18. The apparatus of claim 17, wherein:

said responder processor comprises means for processing the embedded software using a hash function (210) to produce said digest; and

said challenger processor comprises means for processing said copy of the embedded software using said hash function to produce said verification digest.

19. The apparatus of claim 18, wherein:

said challenger transceiver comprises means for transmitting a nonce (204) from said challenger to the responder;

said responder processor comprises means for processing said nonce and the embedded software using said hash function to produce said digest; and

said challenger processor comprises means for processing said nonce and said copy of the embedded software using said hash function to produce said verification digest.

20. The apparatus of claim 19, wherein said nonce is unique.

21. The apparatus of claim 20, wherein

said responder processor comprises:

a catenator (206B) that catenates said nonce and the embedded software to produce a pre-image (208B), and

means for processing said pre-image using said hash function to produce said digest; and

said challenger processor comprises:

a further catenator (206A) that catenates said nonce and said copy of said embedded software to produce a verification pre-image (208A), and

means for processing said verification pre-image using said hash function to produce said verification digest.

22. The apparatus of claim 21, wherein the embedded software includes a unique identifier.

23. The apparatus of claim 22, wherein:

said responder processor comprises means for processing said nonce and the embedded software using a cryptographic hash function to produce said digest; and

said challenger processor comprises means for processing said nonce and said copy of the embedded software using said cryptographic hash function to produce said verification digest.

24. The apparatus of claim 23, wherein said challenger processor comprises a difference element that compares said received digest and said verification digest.

25. The apparatus of claim 24, wherein said means for authenticating comprises means for authenticating the embedded software when said received digest and said verification digest match.

26. The apparatus of claim 25, wherein the embedded software occupies a portion of the memory (108) of the responder, further comprising:

means for populating the remaining portion of the memory with a predetermined bit pattern; and wherein

said means for processing said nonce and the embedded software comprises means for processing said nonce, the embedded software and said predetermined bit pattern; and

said means for processing said nonce and said copy of the embedded software comprises means for processing said nonce, said copy of the embedded software, and said predetermined bit pattern.

27. A method for authenticating embedded software in the memory of a responder over a communications channel, comprising the steps of:

processing the embedded software to produce a digest; and

transmitting said digest to a challenger;

whereby said challenger can authenticate the embedded software using said digest and a verification digest produced by processing a copy of the embedded software.

28. An apparatus for authenticating embedded software in the memory (108) of a responder (102) over a communications channel (130), comprising:

a processor (106) that processes the embedded software to produce a digest (212B); and

a transmitter (104) that transmits said digest to a challenger (122);

whereby said challenger can authenticate the embedded software using said digest and a verification digest (212A) produced by processing a copy of the embedded software.

29. A method for authenticating embedded software in the memory of a responder over a communications channel, comprising the steps of:

receiving a digest from the responder, said digest produced by processing the embedded software;

processing the received digest and a verification digest to produce a result; and

authenticating the embedded software according to said result.

30. An apparatus for authenticating embedded software in the memory (108) of a responder (102) over a communications channel (130), comprising:

a receiver (124) that receives a digest (212B) from the responder, said digest produced by processing the embedded software; and

a processor (126) that processes the received digest and a verification digest (212A) to produce a result (216);

whereby the embedded software is authenticated when said result indicates a match.

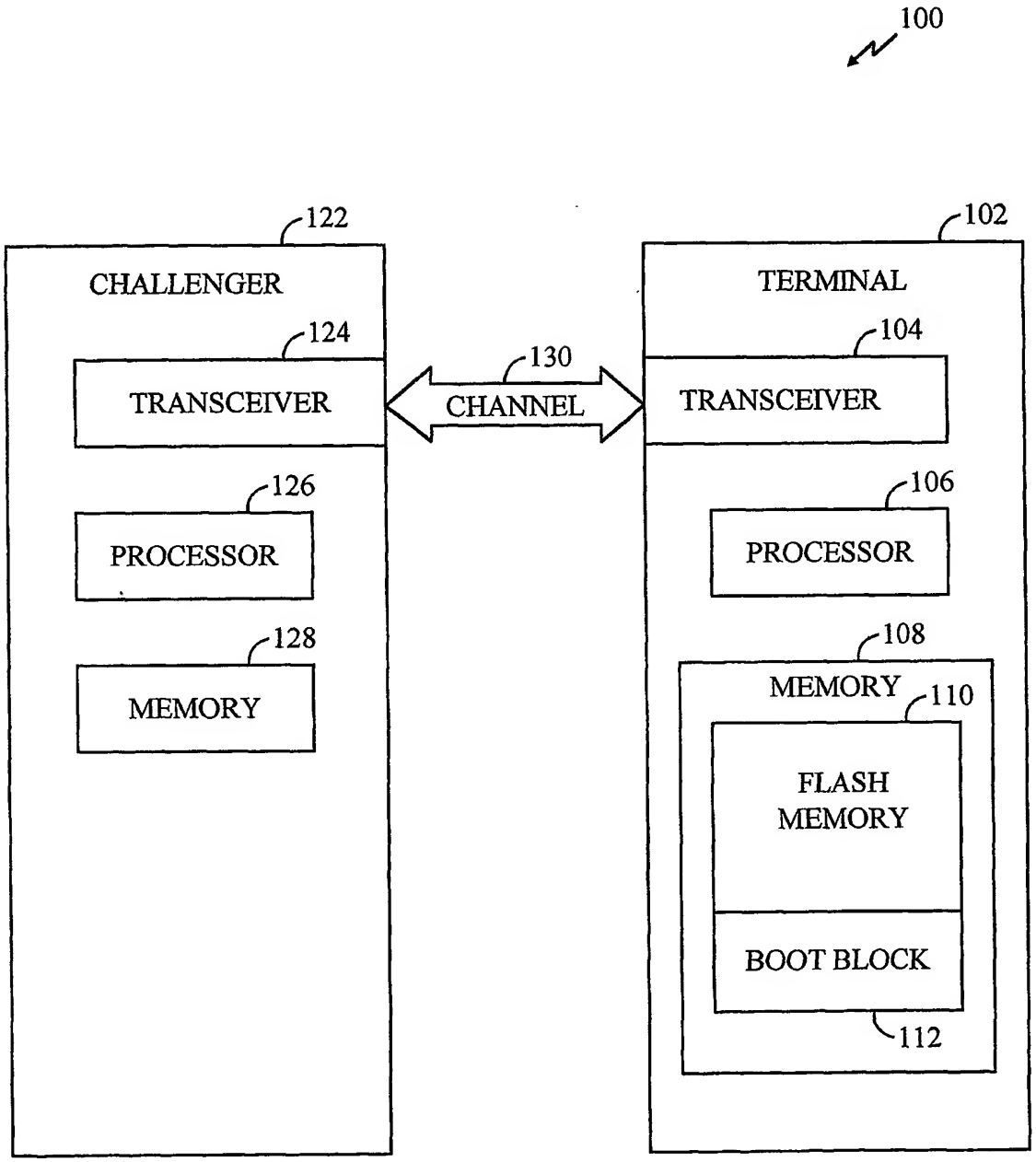

100
122
CHALLENGER
124
TRANSCEIVER
126
PROCESSOR
128
MEMORY
130
CHANNEL
102
TERMINAL
104
TRANSCEIVER
106
PROCESSOR
108
MEMORY
110
FLASH MEMORY
BOOT BLOCK
112

FIG. 1

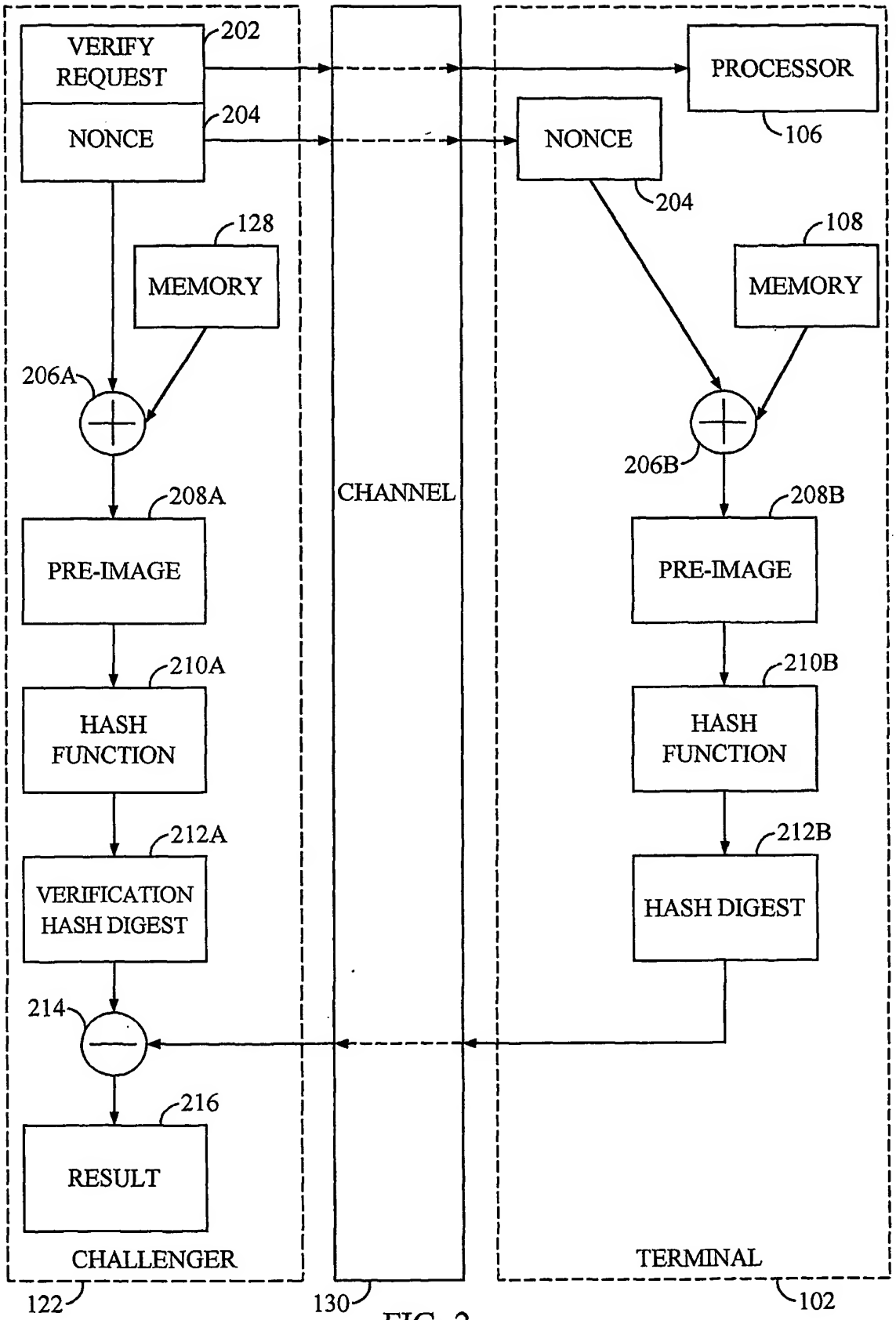

VERIFY REQUEST
202
NONCE
204
128
MEMORY
206A
208A
PRE-IMAGE
210A
HASH FUNCTION
212A
VERIFICATION HASH DIGEST
214
216
RESULT
CHALLENGER
122
CHANNEL
130
PROCESSOR
106
NONCE
204
108
MEMORY
206B
208B
PRE-IMAGE
210B
HASH FUNCTION
212B
HASH DIGEST
TERMINAL
102

FIG. 2

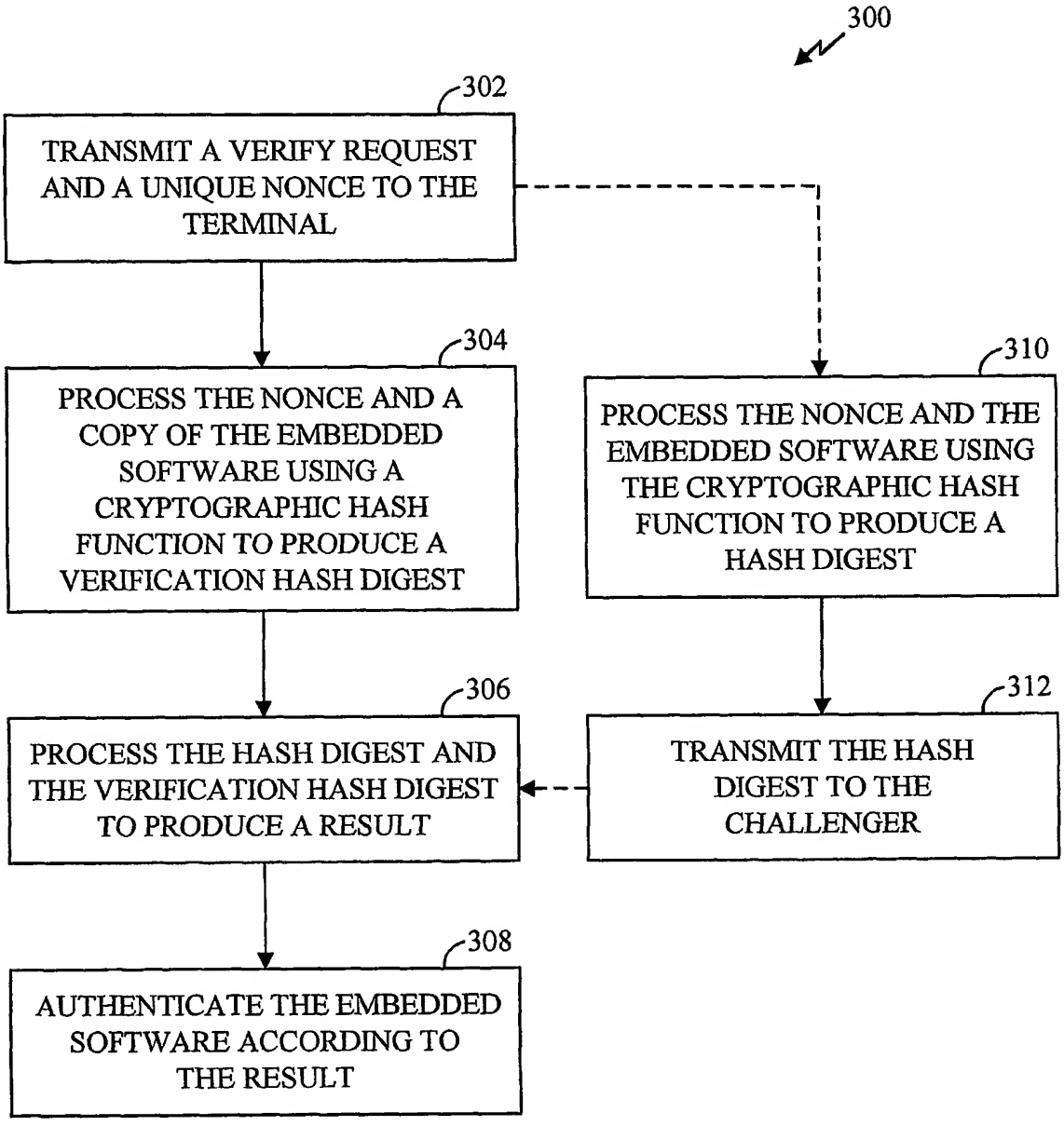

300
302
TRANSMIT A VERIFY REQUEST AND A UNIQUE NONCE TO THE TERMINAL
304
PROCESS THE NONCE AND A COPY OF THE EMBEDDED SOFTWARE USING A CRYPTOGRAPHIC HASH FUNCTION TO PRODUCE A VERIFICATION HASH DIGEST
310
PROCESS THE NONCE AND THE EMBEDDED SOFTWARE USING THE CRYPTOGRAPHIC HASH FUNCTION TO PRODUCE A HASH DIGEST
306
PROCESS THE HASH DIGEST AND THE VERIFICATION HASH DIGEST TO PRODUCE A RESULT
312
TRANSMIT THE HASH DIGEST TO THE CHALLENGER
308
AUTHENTICATE THE EMBEDDED SOFTWARE ACCORDING TO THE RESULT

FIG. 3

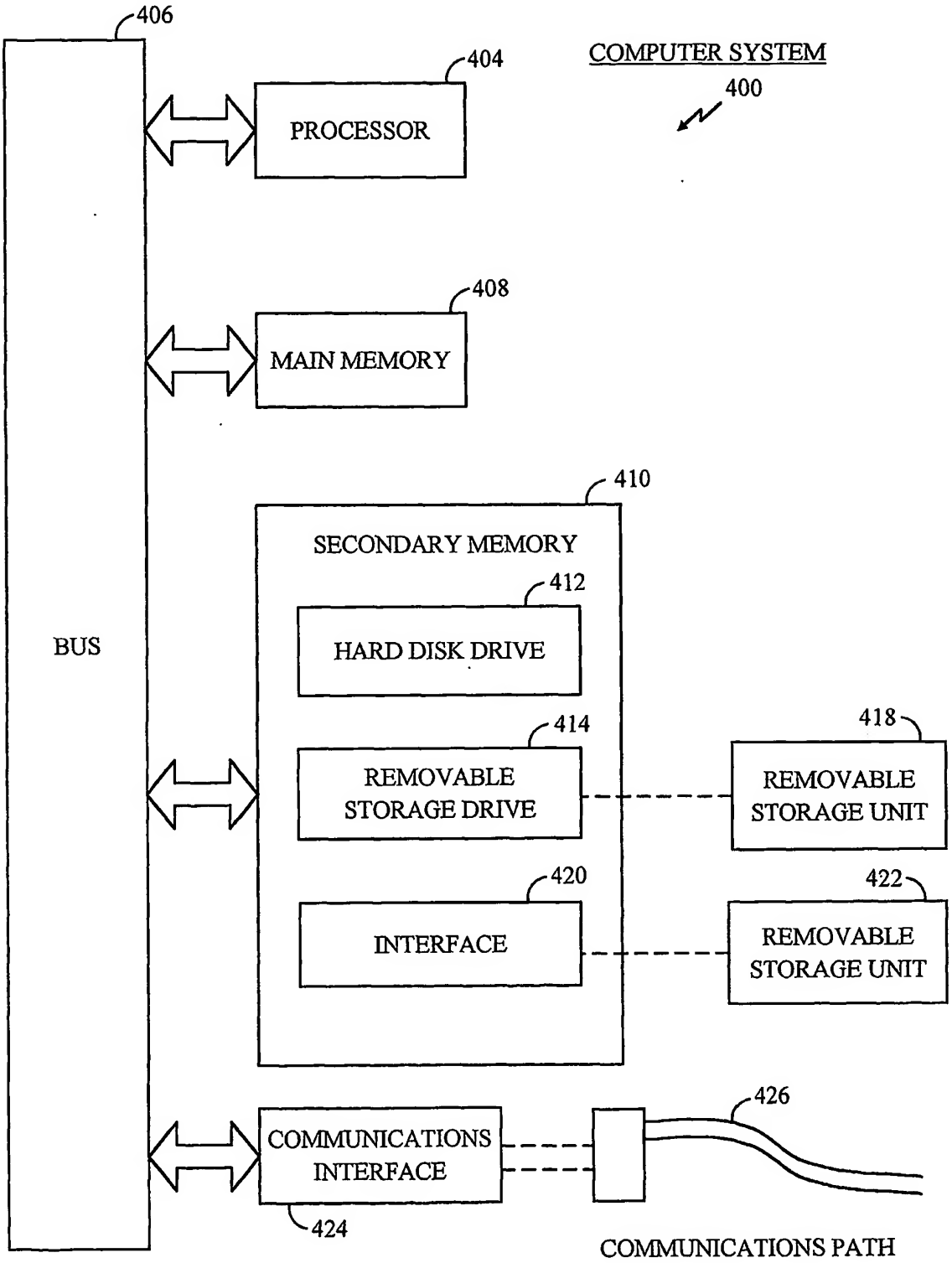

406
COMPUTER SYSTEM
400
404
PROCESSOR
408
MAIN MEMORY
410
SECONDARY MEMORY
412
HARD DISK DRIVE
BUS
414
REMOVABLE STORAGE DRIVE
418
REMOVABLE STORAGE UNIT
420
INTERFACE
422
REMOVABLE STORAGE UNIT
426
COMMUNICATIONS INTERFACE
424
COMMUNICATIONS PATH

FIG. 4